

UNIT 2 – JDBC

JDBC –

- stands for **Java Database Connectivity**.
- standard Java API for database-independent connectivity between the Java programming language, and a wide range of databases.
- Java API which can access any kind of tabular data → especially the data which are stored in a relational database
- Works with Java on a variety of platforms such as → Windows, Mac OS and various versions of Unix
- Primary packages for JDBC 4.0 → java.sql and javax.sql

Use of JDBC Library –

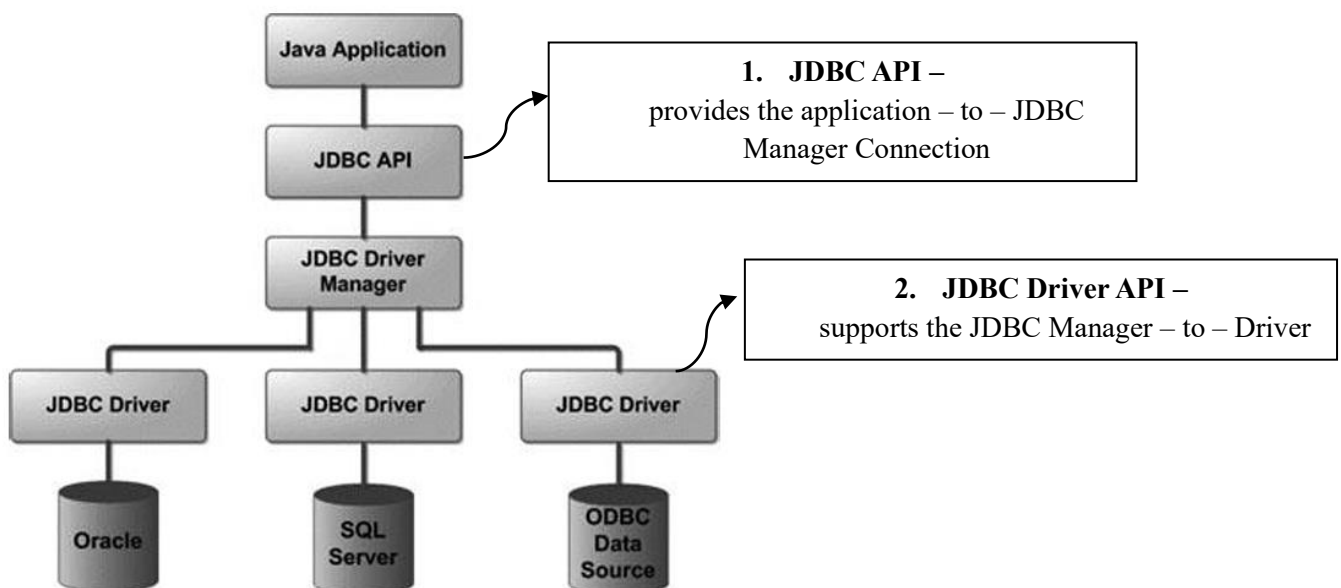
- JDBC library → includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
 1. Making a connection to a database.
 2. Creating SQL or MySQL statements.
 3. Executing SQL or MySQL queries in the database.
 4. Viewing + modifying the resulting records.
- JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.
- Java Executables –
 1. Java Applications
 2. Java Applets
 3. Java Servlets
 4. Java ServerPages (JSPs)
 5. Enterprise JavaBeans (EJBs).

All executables able to use a JDBC Driver to –

1. access a database
2. take advantage of the stored data.

JDBC Architecture –

JDBC Architecture consists of 2 layers –



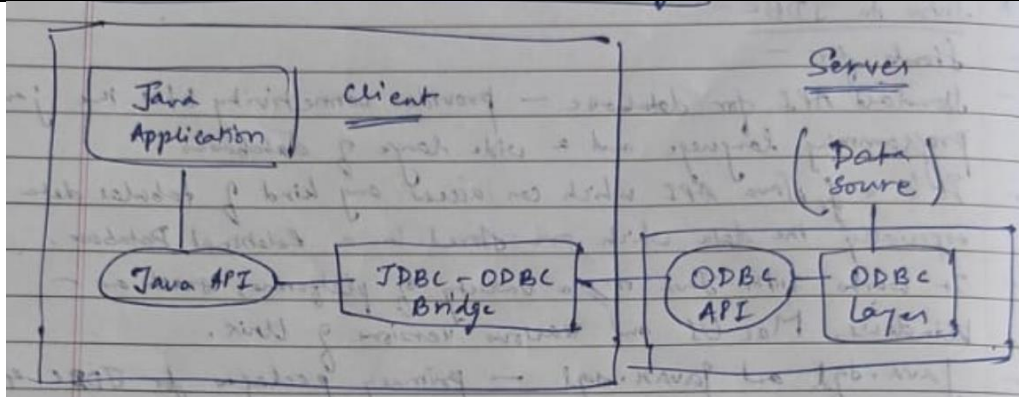
Components of JDBC –

JDBC Component	Type (is a Class / Object / Interface)	Description
1. Driver Manager	Class	<ul style="list-style-type: none">Manages a list of database driversMatches connection requests from the java application with the proper database driver using communication subprotocol. <p style="text-align: center;">↓</p> <p>first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.</p>
2. Driver	Interface	<ul style="list-style-type: none">Handles the communications with the database server.Abstracts the details associated with working with Driver objects.
3. Connection	Interface	<ul style="list-style-type: none">Consists of all methods for contacting a database.Connection object = communication context, i.e., all communication with database is through connection object only.
4. Statement	Interface	<ul style="list-style-type: none">Use objects created from this interface to submit the SQL statements to the database.
5. ResultSet	Objects	<ul style="list-style-type: none">Hold data retrieved from a database after you execute an SQL query using Statement objects.Acts as an iterator to allow you to move through its data.
6. SQLException	Class	<ul style="list-style-type: none">Handles any errors that occur in a database application.

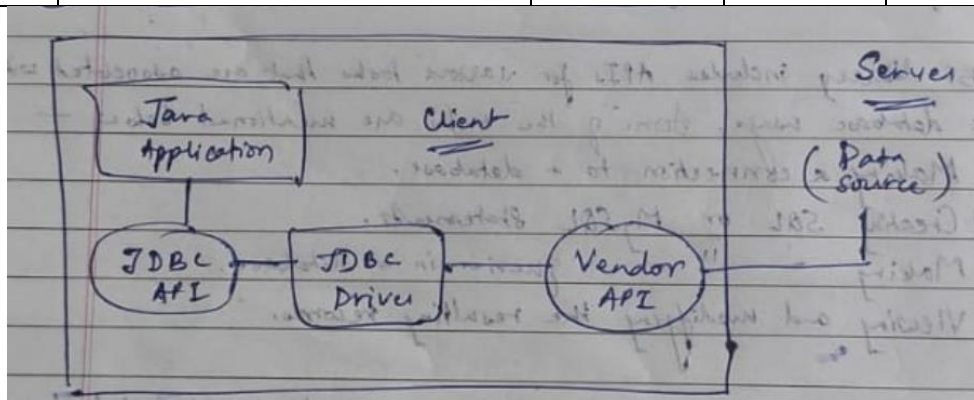
JDBC Drivers –

- Implement the defined interfaces in the JDBC API, for interacting with your database server.
- enable to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- To use – import ‘java.sql’ package OR implement ‘java.sql.Driver’ interface in database driver.
- 4 types of JDBC Drivers –
 - Type 1:** JDBC – ODBC Bridge Driver
 - Type 2:** JDBC – Native API
 - Type 3:** JDBC – Net Pure Java
 - Type 4:** 100% Pure Java

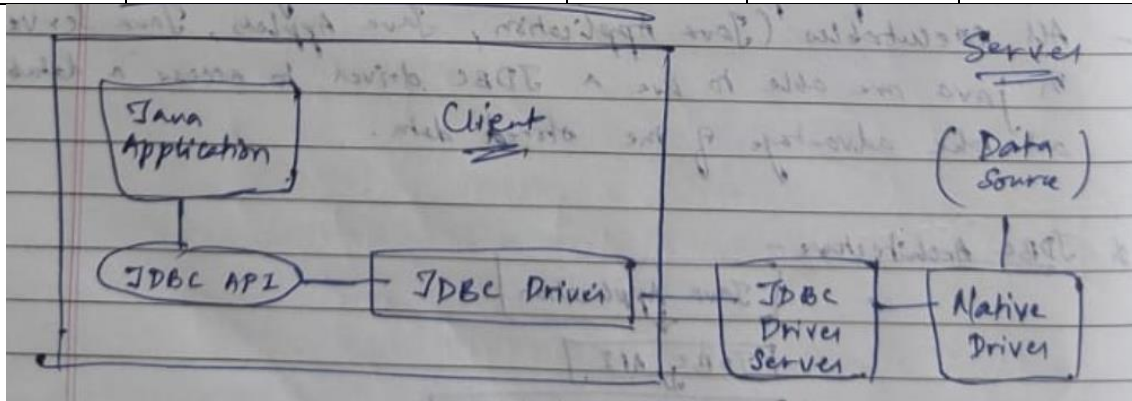
Type of JDBC Driver	Description	Installations required on machine / system	Example	When to use
Type 1: JDBC-ODBC Bridge Driver (ODBC Stands for – Open Database Connectivity)	<ul style="list-style-type: none"> A JDBC bridge → used to access ODBC drivers installed on each client machine. 	Configuration of a Data Source Name (DSN) that represents the target database.	The JDBC-ODBC Bridge that comes with JDK 1.2	<ul style="list-style-type: none"> Not considered a deployment-level driver Used for development and testing purposes only.



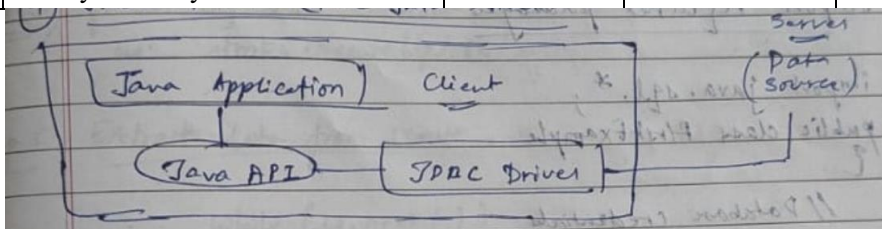
Type 2: JDBC-Native API	<ul style="list-style-type: none"> JDBC API calls are converted into native C/C++ API calls, which are unique to the database. Advantage – high speed due to elimination of ODBC's overhead. Disadvantage – changing the database requires changing the native API since it is specific to the database. 	Vendor-specific driver on each client machine	Oracle Call Interface (OCI) driver	Useful in situations where a type 3 or type 4 driver is not available yet for your database.
-----------------------------------	---	---	------------------------------------	--



<p>Type 3: JDBC-Net Pure Java</p>	<ul style="list-style-type: none"> A three-tier approach JDBC clients use standard network sockets to communicate with a middleware application server. ↓ The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server. Advantages – <ol style="list-style-type: none"> Extremely flexible → requires no code installed on the client A single driver can provide access to multiple databases. 	None	---	If Java application is accessing multiple types of databases at the same time
--	--	------	-----	---



<p>Type 4: 100% Pure Java</p>	<ul style="list-style-type: none"> A pure Java-based driver communicates directly with the vendor's database through socket connection. Advantages – <ol style="list-style-type: none"> Highest performance driver Provided by the vendor itself Extremely flexible – does not require installing any special software on the client / server. Can be downloaded dynamically. 	None	MySQL's Connector / J driver	If accessing one type of database, such as Oracle, Sybase, or IBM
--	---	------	------------------------------------	---



Steps involved in Creating a JDBC Application –

1. **Import the packages** – include the packages containing the JDBC classes needed for database programming → using import java.sql.* ;
2. **Register the JDBC driver** – initialize a driver to open a communication channel with the database.
3. **Open a connection** – use the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with the database.
4. **Execute a query** – use an object of type ‘Statement’ for building and submitting an SQL statement to the database.
5. **Extract data from result set** – use the appropriate ResultSet.getXXX() method to retrieve the data from the result set.
6. **Clean up the environment** – explicitly close all database resources versus relying on the JVM's garbage collection.

CODE TO EXPLAIN THE STEPS INVOLVED –

//STEP 1. Import required packages

```
import java.sql.*;
```

```
public class FirstExample
```

```
{
```

```
    public static void main(String[] args) {
```

```
        Connection conn = null;
```

```
        Statement stmt = null;
```

```
        try
```

```
        {
```

```
            //STEP 2: Register JDBC driver
```

```
            Class.forName("com.mysql.jdbc.Driver");
```

```
            //STEP 3: Open a connection
```

```
            System.out.println("Connecting to database...");
```

```
            conn = DriverManager.getConnection("jdbc:mysql://localhost/EMP", "root", "1234");
```

```
            // Database URL, Username, Password
```

```
            //STEP 4: Execute a query
```

```
            System.out.println("Creating statement...");
```

```
            stmt = conn.createStatement();
```

```
            String sql;
```

```
            sql = "SELECT id, first, last, age FROM Employees";
```

```
            ResultSet rs = stmt.executeQuery(sql);
```

```
            // stmt.executeQuery(sql) → in case of SELECT statement
```

```
            // stmt.executeUpdate(sql) → in case of other statements such as INSERT INTO, UPDATE etc.
```

```
            //STEP 5: Extract data from result set
```

```
            while(rs.next()){
```

```
                //Retrieve by column name
```

```
int id = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");

//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
System.out.println(", Last: " + last);
}
```

//STEP 6: Clean-up environment

```
rs.close();
stmt.close();
conn.close();
}
```

```
catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
}
```

```
catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
}
```

```
}//end main
```

```
}//end FirstExample
```